SYSTEM AND METHOD FOR RETIRING APPROXIMATELY SIMULTANEOUSLY A GROUP OF INSTRUCTIONS IN A SUPERSCALAR MICROPROCESSOR

Inventors: Johannes Wang
Sanjiv Garg
Trevor Deosaran

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application is a continuation of Application Ser. No. 09/631,640, filed August 2, 2000, presently allowed, which is a continuation of Application Ser. No. 09/009,412, filed January 20, 1998, now U.S. Patent No. 6,131,157, which is a continuation of Application Ser. No. 08/481,146 filed June 7, 1995, now U.S. Patent No. 5,826,055, which is a continuation of Application Ser. No. 07/877,451, filed May 1, 1992, now abandoned.

[0002] The following patents are related to the subject matter of the present application and are incorporate by reference in their entirety herein: "Superscalar RISC Instruction Scheduling," U.S. Pat. No. 5,497,499, issued March 5, 1996; and "High Performance, Superscalar-Based Computer System with Out-of-Order Instruction Execution," U.S. Pat. No. 5,539,911, issued July 23, 1996.

BACKGROUND OF THE INVENTION

Field of the Invention

[0003] The present invention relates generally to the field of superscalar processors, and more particularly, to a system and method for retiring multiple instructions executed out-of-order in a superscalar processor.

Discussion of Related Art

[0004] One method of increasing performance of microprocessor-based systems is overlapping the steps of different instructions using a technique called pipelining. In pipelining operations, various steps of instruction execution (e.g. fetch, decode and execute) are performed by independent units called pipeline stages. The steps are performed in parallel in the various pipeline stages so that the processor can handle more than one instruction at a time.

[0005] As a result of pipelining, processor-based systems are typically able to execute more than one instruction per clock cycle. This practice allows the rate of instruction execution to exceed the clock rate. Processors that issue, or initiate execution of, multiple independent instructions per clock cycle are known as superscalar processors. A superscalar processor reduces the average number of cycles per instruction beyond what is possible in ordinary pipelining systems.

[0006] In a superscalar system, the hardware can execute a small number of independent instructions in a single clock cycle. Multiple instructions can be executed in a single cycle as long as there are no data dependencies, procedural dependencies, or resource conflicts. When such dependencies or conflicts exist, only the first instruction in a sequence can be executed. As a result, a plurality of functional units in a superscalar architecture cannot be fully utilized.

[0007] To better utilize a superscalar architecture, processor designers have enhanced processor look-ahead capabilities; that is the ability of the processor to examine instructions beyond the current point of execution in an attempt to find independent instructions for immediate execution. For example, if an instruction dependency or resource conflict inhibits instruction execution, a processor with look-ahead capabilities can look beyond the present instruction, locate an independent instruction, and execute it.

[0008] As a result, more efficient processors, when executing instructions, put less emphasis on the order in which instructions are fetched and more emphasis on the

order in which they are executed. As a further result, instructions are executed out of order.

[0009] For a more in-depth discussion of superscalar processors, see Johnson, *Superscalar Microprocessor Design*, Prentice Hall, Inc. (1991).

[0010] Scenarios occur whereby the execution of the instructions is interrupted or altered, and the execution must be restarted in the correct order. Two such scenarios will be described.

[0011] In a first scenario, during look-ahead operations, many processor designs employ predictive techniques to predict a branch that the program is going to follow in that particular execution. In these systems, the instructions fetched and executed as a result of look-ahead operations are instructions from the branch of code that was predicted. High instruction throughput is achieved by fetching and issuing instructions under the assumption that branches chosen are predicted correctly and that exceptions do not occur. This technique, known as speculative execution, allows instruction execution to proceed without waiting for the completion of previous instructions. In other words, execution of the branch target instruction stream begins before it is determined whether the conditional branch will be taken.

[0012] Since the branch prediction occasionally fails, the processor must provide recovery mechanisms for canceling the effects of instructions that were speculatively executed. The processor must also provide restart mechanisms to reestablish the correct instruction sequence.

[0013] In a second scenario, out-of-order completion makes it difficult to deal with exceptions. Exceptions are created by instructions when the instruction cannot be properly executed by hardware alone. These exceptions are commonly handled by interrupts, permitting a software routine to correct the situation. Once the routine is completed, the execution of the interrupted program must be restarted so it can continue as before the exception.

[0014] Processors contains information that must be saved for a program to be suspended and then restored for execution to continue. This information is known

as the "state" of the processor. The state information typically includes a program counter (PC), an interrupt address register (IAR), and a program status register (PSR); the PSR contains status flags such as interrupt enable, condition codes, and so forth.

[0015] As program instructions are executed, the state machine is updated based on the instructions. When execution is halted and must later be restarted (i.e., one of the two above scenarios occurs) the processor looks to the state machine for information on how to restart execution. In superscalar processors, recovery and restart occur frequently and must be accomplished rapidly.

[0016] In some conventional systems, when instructions are executed out of order, the state of the machine is updated out of order (i.e., in the same order as the instructions were executed). Consequently, when the processor goes back to restart the execution, the state of the machine has to be "undone" to put it back in a condition such that execution may begin again.

[0017] To understand conventional systems, it is helpful to understand some common terminology. An in-order state is made up of the most recent instruction result assignments resulting from a continuous sequence of executed instructions. Assignments made by instructions completed out-of-order where previous instruction(s) have not been completed, are not included in this state.

[0018] If an instruction is completed and all previous instructions have also been completed, the instruction's results can be stored in the in-order state. When instructions are stored in the in-order state, the machine never has to access results from previous instructions and the instruction is considered "retired."

[0019] A look-ahead state is made up of all future assignments, completed and uncompleted, beginning with the first uncompleted instruction. Since there are completed and uncompleted instructions, the look-ahead state contains actual as well as pending register values.

[0020] Finally, an architectural state is made up of the most recently completed assignment of the continuous string of completed instructions and all pending assignments to each register. Subsequent instructions executed out of order must

access the architectural state to determine what state the register would be in had the instruction been executed in order.

[0021] One method used in conventional systems to recover from misdirected branches and exceptions is known as checkpoint repair. In checkpoint repair, the processor provides a set of logical spaces, only one of which is used for current execution. The other logical spaces contain backup copies of the in-order state, each corresponding to a previous point in execution. During execution, a checkpoint is made by copying the current architectural state to a backup space. At this time, the oldest backup state is discarded. The checkpoint is updated as instructions are executed until an in-order state is reached. If an exception occurs, all previous instructions are allowed to execute, thus bringing the checkpoint to the in-order state.

[0022] To minimize the amount of required overhead, checkpoints are not made at every instruction. When an exception occurs, restarting is accomplished by loading the contents of the checkpointed state preceding the point of exception, and then executing the instructions in order up to the point of exception. For branch misprediction recovery, checkpoints are made at every branch and contain the precise state at which to restart execution immediately.

[0023] The disadvantage of checkpoint repair is that it requires a tremendous amount of storage for the logical spaces. This storage overhead requires additional chip real estate which is a valuable and limited resource in the microprocessor.

[0024] Other conventional systems use history buffers to store old states that have been superseded by new states. In this architecture, a register buffer contains the architectural state. The history buffer is a last-in first-out (LIFO) stack containing items in the in-order state superseded by look-ahead values (i.e., old values that have been replaced by new values), hence the term "history."

[0025] The current value (prior to decode) of the instruction's destination register is pushed onto the stack. The value at the bottom of the stack is discarded if its associated instruction has been completed. When an exception occurs, the processor suspends decoding and waits until all other pending instructions are

completed, and updates the register file accordingly. All values are then popped from the history buffer in LIFO order and written back into the register file. The register file is now at the in-order state at the point of exception.

[0026] The disadvantage associated with the history buffer technique is that several clock cycles are required to restore the in-order state.

[0027] Still other conventional systems use a reorder buffer managed as a first-in first-out (FIFO) queue to restart after exceptions and mispredictions. The reorder buffer contains the look-ahead state, and a register file contains the in-order state. These two can be combined to determine the architectural state. When an instruction is decoded, it is assigned an entry at the top of the reorder buffer. When the instruction completes, the result value is written to the allocated entry. When the value reaches the bottom of the buffer, it is written into the register file if there are no exceptions. If the instruction is not complete when it reaches the bottom, the reorder buffer does not advance until the instruction completes. When an exception occurs, the reorder buffer is discarded and the in-order state is accessed.

[0028] The disadvantage of this technique is that it requires associative lookup to combine the in-order and look-ahead states. Furthermore, associative lookup is not straightforward since it must determine the most recent assignments if there is more than one assignment to a given register. This requires that the reorder buffer be implemented as a true FIFO, rather than a more simple, circularly addressed register array.

[0029] What is needed then is a system and method for maintaining a current state of the machine and for efficiently updating system registers based on the results of instructions executed out of order. This system and method should use a minimum of chip real estate and power and should provide quick recovery of the state of the machine up to the point of an exception. Furthermore, the system should not require complex steps of associative lookup to obtain the most recent value of a register.

SUMMARY OF THE INVENTION

[0030] The present invention is a system and method for retiring instructions issued out of order in a superscalar microprocessor system. According to the technique of the present invention, results of instructions executed out of order are first stored in a temporary buffer until all previous instructions have been executed. Once all previous instructions have been executed and their results stored in order in a register array, the results of the instruction in question can be written to the register array and the instruction is considered retired.

[0031] The register array contains the current state of the machine. To maintain the integrity of register array data, only results of instructions are not written to the register array until the results of all previous instructions have been written. In this manner, the state of the machine is updated in order, and situations such as exceptions and branch mispredictions can be handled quickly and efficiently.

[0032] The present invention comprises means for assigning and writing instruction results to a temporary storage location, transferring results from temporary storage to the register array so that the register array is updated in an in-order fashion and accessing results in the register array and temporary storage for subsequent operations.

[0033] Further features and advantages of the present invention, as well as the structure and operation of various embodiments of the present invention, are described in detail below with reference to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS/FIGURES

[0034] FIG. 1 is a data path diagram of a superscalar instruction execution unit.

[0035] FIG. 2 is a block diagram illustrating the functions of the superscalar instruction execution unit.

[0036] FIG. 3 is a diagram further illustrating the instruction FIFO and the instruction window.

[0037] FIG. 4 is a diagram illustrating instruction retirement according to the present invention.

[0038] FIG. 5A shows the configuration of an instruction window.

[0039] FIG. 5B is a diagram illustrating the assignment of instruction results to storage locations in a temporary buffer according to the present invention.

[0040] FIG. 6A is a timing diagram illustrating data writing to a register array according to the present invention.

[0041] FIG. 6B is a timing diagram illustrating writing results to four register locations per clock cycle according to the present invention.

[0042] In the drawings, like reference numbers indicate identical or functionally similar elements. Additionally, the left-most digit of a reference number identifies the drawing in which the reference number first appears.

DETAILED DESCRIPTION OF THE INVENTION

1. Overview

[0043] The present invention provides a system and a method for retiring completed instructions such that to the program it appears that the instructions are executed sequentially in the original program order. The technique of the present invention is to store all out-of-order instruction results (results of instructions not executed in the program order) in a temporary buffer until all previous instructions are complete without any exceptions. The results are then transferred from the temporary buffer to a register array which represents the official state.

[0044] When an instruction is retired, all previous instructions have been completed and the retired instruction is officially completed. When instructions are retired according to the technique of the present invention, the state of the machine is updated in order. Therefore, when an exception occurs, out-of-order execution is suspended and all uncompleted instructions prior to the exception are executed and retired. Thus, the state of the machine is up to date as of the time of

the exception. When the exception is complete, out-of-order execution resumes from the point of exception. When a branch misprediction is detected, all instructions prior to the branch are executed and retired, the state of the machine is now current, and the machine can restart at that point. All results residing in the temporary buffer from instructions on the improper branch are ignored. As new instructions from the correct branch are executed, their results are written into the temporary buffer, overwriting any results obtained from the speculatively executed instruction stream.

2. Environment

[0045] FIG. 1 illustrates a block diagram of a superscalar Instruction Execution Unit (IEU) capable of out-of-order instruction issuing. Referring to FIG. 1, there are two multi-ported register files 102A, 102B which hold general purpose registers. Each register file 102 provides five read ports and two write ports. Each write port allows two writes per cycle. In general, register file 102A holds only integer data while register file 102B can hold both floating point and integer data.

[0046] Functional units 104 are provided to perform processing functions. In this example, functional units 104 are three arithmetic logic units (ALUs) 104A, a shifter 104B, a floating-point ALU 104C, and a floating-point multiplier 104D. Floating-point ALU 104C and floating-point multiplier 104D can execute both integer and floating-point operations.

[0047] Bypass multiplexers 106 allow the output of any functional unit 104 to be used as an input to any functional unit 104. This technique is used when the results of an instruction executed in one clock cycle are needed for the execution of another instruction in the next clock cycle. Using bypass multiplexers 106, the result needed can be input directly to the appropriate functional unit 104. The instruction requiring those results can be issued on that same clock cycle. Without bypass multiplexers 106, the results of the executed instruction would have to be written to register file 102 on one clock cycle and then be output to the functional

unit 104 on the next clock cycle. Thus, without bypass multiplexers 106 one full clock cycle is lost. This technique, also known as forwarding, is well known in the art and is more fully described in Hennessy et al., *Computer Architecture a Quantitative Approach*, Morgan Kaufmann Publishers (1990) on pages 260-262.

[0048] Selection multiplexers 108 provide a means for selecting the results from functional units 104 to be written to register files 102.

[0049] FIG. 2 illustrates a block diagram of IEU control logic 200 for the IEU shown in FIG. 1. IEU control logic 200 includes an instruction window 202. Instruction window 202 defines the instructions which IEU control logic 200 may issue during one clock cycle. Instruction window 202 represents the bottom two locations in an instruction buffer, which is a FIFO register containing instructions to be executed. This instruction buffer is also referred to as an instruction FIFO. As instructions are completed, they are flushed out at the bottom and new instructions are dropped in at the top. The bottom location of instruction window 202 is referred to as bucket 0 and the top location of instruction window 202 is referred to as bucket 1.

[0050] When all four instructions in bucket 0 have been retired, they are flushed out of bucket 0, the instructions in bucket 1 drop into bucket 0 and a new group of four instructions drops into bucket 1. Instruction window 202 may be implemented using a variety of techniques. One such technique is fully described in U.S. Patent No. 5,497,499, entitled "Superscalar RISC Instruction Scheduling" and issued March 5, 1996, the disclosure of which is incorporated herein by reference.

[0051] In the current example, instruction window 202 contains eight instructions. Therefore, IEU control logic 200 tries to issue a maximum number of instructions from among these eight during each clock cycle. Instruction decoding occurs in decoders 203. Instruction decoding is an ongoing process performed in IEU control logic 200. Instructions must be decoded before dependency checking (discussed below), issuing and execution occur.

[0052] IEU control logic 200 also contains register renaming circuitry (RRC) 204 which performs two related functions. The first function performed is data dependency checking. Once data dependency checking is complete, RRC 204 assigns tags to each instruction which are used to track the location of instruction operands and results.

[0053] Data dependency checking logic, residing in RRC 204, is used for checking instructions for dependencies. In checking for dependencies, the data dependency checking logic looks at the various register file source and destination addresses to determine whether one or more previous instructions must be executed before a subsequent instruction may be executed. FIG. 3 further illustrates instruction window 202 and the instruction FIFO. Referring to FIG. 3, various register file source and destination addresses 302 of the instruction I0 must be checked against the source and destination addresses of all other instructions.

[0054] Referring back to FIG. 2, since instruction window 202 in this example can contain 8 instructions, the IEU can look at eight instructions for scheduling purposes. All source register addresses must be compared with all previous destination addresses. If one instruction is dependent upon completion of a previous instruction, these two instructions cannot be completed out of order. In other words, if instruction I2 requires the results of instruction I1, a dependency exists and I1 must be executed before I2. Some instructions may be long-word instructions, which require extra care when checking for dependencies. For long-word instructions, the instructions occupy two registers both of which must be checked when examining this instruction for dependencies.

[0055] An additional function performed in RRC 204 is tag assignment. Proper tag assignment is crucial to effective instruction retirement according to the present invention. Each instruction in instruction window 202 is assigned a tag based on its location in instruction window 202, and based on the results of data dependency checking discussed above. The tag assigned to each instruction indicates where in a temporary buffer that instruction's results are to be stored until that instruction is retired and whether all of the previous instructions on

which that instruction is dependent have been completed. Tag assignment and the temporary buffer are discussed in more detail below.

[0056] A further function performed by IEU control logic 200 is determining which instructions are ready for issuing. An instruction issuer 208 issues instructions to the appropriate functional unit 104 for execution. Circuitry within RRC 204 determines which instructions in instruction window 202 are ready for issuing and sends a bit map to instruction issuer 208 indicating which instructions are ready for issuing. Instruction decode logic 203 indicates the resource requirement for each instruction. Issuer 208 also receives information from functional units 104 concerning resource availability. This information is scanned by issuer 208 and an instruction is selected for issuing.

[0057] Instruction issuer 208 sends a control signal 209 to multiplexers 210 telling them which instruction to send to functional units 104. Instruction issuer 208 also sends a control signal 211 to multiplexer 212 configuring it to send the appropriate register address to configure the register that is to receive the results of the instruction. Depending on the availability of functional units 104, issuer 208 may issue multiple instructions each clock cycle.

[0058] Referring again to FIGS. 1 and 2, once an instruction is issued to functional units 104 and executed by the same, register files 102A and 102B must be updated to reflect the current state of the machine. When the machine has to "go back" and restart an execution because of an exception or a branch misprediction, the state of the machine must reflect the up-to-date state at the time the exception or branch occurred. Even when instructions are issued and executed out of order, the state of the machine must still reflect, or be recoverable to, the current state at the time of exception or branching.

[0059] The Instruction Retirement Unit (IRU) of the present invention, retires the instructions as if they were executed in order. In this manner, the state of the machine is updated, in order, to the point of the most recent instruction in a sequence of completed instructions.

[0060] The present invention provides a unique system and method for retiring instructions and updating the state of the machine such that when a restart is required due to an exception or a branch misprediction, the current state up to that point is recoverable without needing to wait for the register file to be rebuilt or reconstructed to negate the effects of out-of-order executions.

3. Implementations

[0061] FIG. 4 illustrates a high-level diagram of an Instruction Retirement Unit 400 (referred to as "IRU 400") of the present invention. IRU 400 and its functions are primarily contained within register file 102 and a retirement control block (RCB) 409. As shown in FIG. 4, the functions performed by the environment are also critical to proper instruction retirement.

[0062] Referring to FIG. 4, the operation of IRU 400 will now be described. As discussed in subsection 2 of this application, the instructions executed in the superscalar processor environment are executed out of order, and the out-of-order results cannot be written to the registers until all previous instructions' results are written in order. A register array 404 represents the in-order state of the machine. The results of all instructions completed without exceptions, who also have no previous uncompleted instructions, are stored in register array 404. Once the results are stored in register array 404, the instruction responsible for those results is considered "retired."

[0063] If an instruction is completed out of order, and there are previous instructions that have not been completed, the results of that instruction are temporarily stored in a temporary buffer 403. Once all instructions previous to the instruction in question have been executed and their results transferred to register array 404, the instruction in question is retirable, and its results can be transferred from temporary buffer 403 to register array 404. Once this is done, the instruction is considered retired. A retirable instruction then, is an instruction for which two

conditions have been met: (1) it is completed, and (2) there are no unexecuted instructions appearing earlier in the program order.

[0064] If the results of an executed instruction are required by a subsequent instruction, those results will be made available to the appropriate functional unit 104 regardless of whether they are in temporary buffer 403 or register array 404.

[0065] Referring to FIGS. 1, 2, and 4, IRU 400 will be more fully described. Register file 102 includes a temporary buffer 403, a register array 404 and selection logic 408. There are two input ports 110 used to transfer results to temporary buffer 403 and register array 404. Control signals (not shown) generated in IEU control logic 200 are used to select the results in selection multiplexer 108 when the results are ready to be stored in register file 102. Selection multiplexer 108 receives data from various functional units and multiplexes this data onto input ports 110.

[0066] Two input ports 110 for each register file 102 in the preferred embodiment permit two simultaneous register operations to occur. Thus, input ports 110 provide two full register width data values to be written to temporary buffer 403. This also permits multiple register locations to be written in one clock cycle. The technique of writing to multiple register address locations in one clock cycle is fully described below.

[0067] FIGS. 5A and B illustrate the allocation of temporary buffer 403. FIG. 5A shows a configuration of instruction window 202, and FIG. 5B shows an example ordering of data results in temporary buffer 403. As noted previously, there can be a maximum of eight pending instructions at any one time. Each instruction may require one or two of temporary buffer's 403 eight register locations 0 through 7, depending on whether it is a regular-length or a long-word instruction.

[0068] The eight pending instructions in instruction window 202 are grouped into four pairs. The first instructions from buckets 0 and 1 (i.e. I0 and I4) are a first pair. The other pairs, I1 and I5, etc., are similarly formed. A result of I0 (I0RD) is stored in register location 0, and a result of I4 (I4RD) is stored in register location 1. If I0 is a long-word entry, I0RD, the low-word result (result of the first

half of a long-word instruction) is still stored in location 0, but now the high-word result (I0RD+1, from the second half of the instruction) is stored in location 1. This means that the low-word result of I4 does not have a space in temporary buffer 403, and therefore can not be issued at this time.

[0069] Tags are generated in RRC 204 and assigned to each instruction before the instruction's results are store in temporary buffer 403. This facilitates easy tracking of results, particularly when instructions are executed out of order. Each tag comprises three bits, for example, to indicate addresses for writing the instruction's results in temporary buffer 403. These three bits are assigned according to the instructions' locations in instruction window 202. The tags are used by the RRC to locate results in temporary buffer 403 if they are operands for other instructions, for example. Table 1 illustrates a representative assignment for these three tag bits.

INSTRUCTION	TAG	LOCATION
0	000	0
1	010	2
2	100	4
3	110	6
4	001	1
5	011	3
6	101	5
7	111	7

Table 1. Tag Assignment

[0070] Each location in instruction window 202 has a corresponding location in temporary buffer 403. The least significant bit indicates the bucket in instruction window 202 where the instructions originated. This bit is interpreted differently

when the bucket containing the instruction changes. For example, when all four instructions of bucket 0 are retired, the instructions in bucket 1 drop into bucket 0. When this occurs the LSB (least significant bit) of the tag that previously indicated bucket 1, now indicates bucket 0. For example, in Table 1, an LSB of 1 indicates the instructions in bucket 1. When these instructions are dropped into bucket 0, the LSB will not change and an LSB of 1 will indicate bucket 0. The tag contains information on how to handle each instruction.

[0071] When the instruction is executed and its results are output from a functional unit, the tag follows. Three bits of each instruction's tag uniquely identify the register location where the results of that instruction are to be stored. A temporary write block (not shown) looks at functional units 104, the instruction results and the tags. Each functional unit 104 has 1 bit that indicates if a result is going to be output from that functional unit 104 on the next clock cycle. The temporary write block gets the tag for each result that will be available on the next clock cycle. The temporary write block generates an address (based on the tag) where the upcoming results are to be stored in temporary buffer 403. The temporary write block addresses temporary buffer 403 via RRC 204 on the next clock cycle when the results are ready at functional unit 104.

[0072] As noted above, a function of the tags is to permit the results of a particular functional unit 104 can be routed directly to the operand input of a functional unit 104. This occurs when a register value represents an operand that is needed immediately by a functional unit 104. The results can also be stored in register array 404 or temporary buffer 403.

[0073] In addition, the tags indicate to the IEU when to return those results directly to bypass multiplexers 106 for immediate use by an instruction executing in the very next clock cycle. The instruction results may be sent to either the bypass multiplexers 106, register file 102, or both.

[0074] The results of all instructions executed out of order are stored first in a temporary buffer 403. As discussed above, temporary buffer 403 has eight storage locations. This number corresponds to the size of instruction window 202. In the

example discussed above, instruction window 202 has eight locations and thus there are up to eight pending instructions. Consequently, up to eight instruction results may need to be stored in temporary buffer 403.

[0075] If an instruction is completed in order, that is all previous instructions are already completed and their results written to register array 404, the results of that instruction can be written directly to register array 404. RCB 409 knows if results can go directly to register array 404. In this situation, RCB 409 sets an external write bit enabling a write operation to register array 404. Note, in the preferred embodiment, the results in this situation are still written to temporary buffer 403. This is done for simplicity.

[0076] For each instruction result in temporary buffer 403, when all previous instructions are complete, without any exceptions or branch mispredictions, that result is transferred from temporary buffer 403 to a register array 404 via selection logic 408. If an instruction is completed out of order and previous instructions are not all completed, the results of that instruction remain in temporary buffer 403 until all previous instructions are completed. If one or more instructions have been completed, and they are all awaiting completion of an instruction earlier in the program order, they cannot be retired. However, once this earlier instruction is completed, the entire group is retirable and can be retired.

[0077] A done block 420 is an additional state machine of the processor. Done block 420 keeps track of what instructions are completed and marks these instructions `done` using a done flag. The done block informs a retirement control block 409 which instructions are done. The retirement control block 409, containing retirement control circuitry checks the done flags to see if all previous instructions of each pending instruction are completed. When retirement control block 409 is informed that all instructions previous (in the program order) to the pending instruction are completed, the retirement control block 409 determines that the pending instruction is retirable.

[0078] FIG. 6A is a timing diagram illustrating writing to register array 404, and FIG. 6B is a timing diagram illustrating the transfer of data from temporary buffer

403 to register array 404. Referring to FIGS. 4, 6A, and 6B, the technique of writing to register array 404 will be described.

[0079] Temporary buffer 403 has four output ports F, G, H, and I that are used to transfer data to register array 404. Register array 404 has two input ports, A' and B', for accepting instruction results from either temporary buffer 403 or functional units 104. Write enable signals 602 and 604 enable writes to temporary buffer 403 and register array 404, respectively, as shown at 603. Although not illustrated, there are actually 2 write enable signals 604 for register array 404. One of these enable signals 604 is for enabling writes to input port A', and the other is for enabling writes to input port B'. Since there are two input ports A', and B', two writes to register array 404 can occur simultaneously.

[0080] Data to be written to register array 404 can come from either temporary buffer 403 or functional units 104 (via selection multiplexer 108 and bus 411). Control signal 606 is used to select the data in selection logic 408. When control signal 606 is a logic high, for example, data is selected from temporary buffer 403. Signal 410 is the write address, dictating the location where data is to be written in either temporary buffer 403 or register array 404. Data signal 608 represents the data being transferred from temporary buffer 403 to register array 404. Alternatively, data signal 608 represents data 110 from functional units 104 via selection multiplexer 108.

[0081] Register array 404 can write 4 locations in one clock cycle. Address 410 and write enable 604 signals are asserted first, then data 608 and control signal 606 are asserted. Control signal 606 is asserted as shown at 605. During the first half of the cycle, registers corresponding to instructions I0 and I1 will be updated. During the second half of the cycle, registers corresponding to I2 and I3 will be updated. If any of the results are long words, the upper half of the word will be updated during the second cycle. Thus, two results can be simultaneously transferred and two instructions can be simultaneously retired in a half a clock cycle. A total of four instructions can therefore be retired per clock cycle.

[0082] Referring to FIG. 6B, read addresses 612F, 612G, 612H, and 6121 are available for temporary buffer 403 output ports F through I. Data 614F, 614G, 614H, and 614I is available from temporary buffer 403 at the beginning of the clock cycle, as shown at 615. Addresses 410A are generated for input port A' and 410B are generated for input port B'. Similarly, a write enable signal 604A for input port A' and a write enable signal 604B for input port B' are generated for each half of the clock cycle. Address 410 appearing in the first half of the clock cycle, as shown at 611A and 611B, is the location to which data is written during enable signal 604 appearing in the first half, as shown as 605A and 605B. Similarly, data is written during the second half of the clock cycle to the address 410 appearing at that time, as shown at 613A and 613B. Since data is written to A' and B' simultaneously, up to four instruction results may be written to register array 404 during one clock cycle. Therefore, up to four instructions may be retired during one clock cycle.

[0083] Latches in selection logic 408 hold the data constant until the appropriate address 410 is present and write enable signals 604 allow the data to be written.

[0084] The process of transferring a result from temporary buffer 403 to register array 404, as described above, is called retiring. When an instruction is retired, it can be considered as officially completed. All instructions previous to that instruction have been completed without branch mispredictions or exceptions and the state of the machine will never have to be redetermined prior to that point. As a result, to the program running in the processor, it appears that the instructions are updated and executed sequentially.

[0085] Since instructions are being issued and executed out of order, subsequent instructions may require operands corresponding to results (values) in temporary buffer 403 or register array 404. Therefore, access to register values in temporary buffer 403, as well as values stored in register array 404 is provided by the present invention.

[0086] Read access to temporary buffer 403 and register file 404 is controlled by RRC 204. Such read access is required by instructions executing that need results

of previously executed instructions. Recall from the discussion in subsection 2 above that RRC 204 performs data dependency checking. RRC 204 knows which instructions are dependent on which instructions and which instructions have been completed. RRC 204 determines if the results required by a particular instruction must be generated by a previous instruction, i.e. whether a dependency exists. If a dependency exists, the previous instruction must be executed first. An additional step is required, however, when a dependency exists. This step is determining where to look for the results of the instruction. Since RRC 204 knows what instructions have been completed, it also knows whether to look for the results of those instructions in temporary buffer 403 or register array 404.

[0087] RRC 204 sends a port read address 410 to register array 404 and temporary buffer 403 to read the data from the correct location onto output lines 412. One bit of read address 410 indicates whether the location is in temporary buffer 403 or register array 404. Again, see U.S. Patent No. 5,497,499, entitled "Superscalar RISC Instruction Scheduling" and issued March 5, 1996 for additional disclosure pertaining to the RRC.

[0088] In the preferred embodiment of the present invention, each output port A through E of temporary buffer 403 and register array 404 has its own dedicated address line. That is, each memory location can be output to any port.

4. Additional Features of the Invention

[0089] IRU 200 also informs other units when instructions are retired. IRU 200 informs an Instruction Fetch Unit (IFU) when it (the IRU) has changed the state of the processor. In this manner, the IFU can maintain coherency with IEU 100. The state information sent to the IFU is the information required to update the current Program Counter and to request more instructions from the IFU. In the example above, when four instructions are retired, the IFU can increment the PC by four and fetch another bucket of four instructions.

[0090] An example of the IFU is disclosed in a commonly owned, copending application Seri. No. 07/817,810 titled "High Performance RISC Microprocessor Architecture."

[0091] In addition, according to a preferred embodiment of the present invention, status bits and condition codes are retired in order as well. Each of the eight instructions in instruction window 202 has its own copy of the status bits and condition codes. If an instruction does not affect any of the status bits, then it propagates the status bits from the previous instruction.

[0092] When an instruction is retired, all its status bits have to be officially updated. If more than one instruction is retired in one cycle, the status bits of the most recent (in order) instruction are used for the update.

5. Conclusion

[0093] While various embodiments of the present invention have been described above, it should be understood that they have been presented by way of example only, and not limitation. Thus, the breadth and scope of the present invention should not be limited by any of the above-described exemplary embodiments, but should be defined only in accordance with the following claims and their equivalents.